Strong Equivalence of RASP Programs

Stefania Costantini[1], Andrea Formisano[2], and David Pearce[3]

[1] Università di L'Aquila, Italy stefania.costantini@univaq.it
[2] Università di Perugia, Italy formis@dmi.unipg.it
[3] Universidad Politècnica de Madrid, Spain david.pearce@upm.es

Abstract. RASP is a recent extension of Answer Set Programming that permits declarative specification and reasoning on consumption and production of resources. In this paper, we extend the concept of strong equivalence (which, as widely recognized, provides an important conceptual and practical tool for program simplification, transformation and optimization) from ASP to RASP programs and discuss its usefulness in this wider context.

Introduction

Answer Set Programming (ASP) is a paradigm of logic programming based on the answer set semantics [1], where solutions to a given problem are represented in terms of selected models (answer sets) of the corresponding logic program [2]. ASP is nowadays applied in many areas, including problem solving, configuration, information integration, security analysis, agent systems, semantic web, and planning (see, among many others, [3, 4]).

In recent work [5, 6, 7], an extension to ASP, called RASP (standing for ASP with Resources), has been proposed so as to support declarative reasoning on consumption and production of resources. RASP is a proper extension to ASP, in the sense that useful theoretical and practical results developed for ASP can be extended to RASP. In this paper in fact, we extend the concept of strong equivalence to RASP programs and discuss its usefulness.

Strong equivalence [8, 9], as widely recognized, provides an important conceptual and practical tool for program simplification, transformation and optimization. Even in the case where two theories are formulated in the same vocabulary, they may have the same answer sets yet behave very differently once they are embedded in some larger context. For a robust or modular notion of equivalence one should require that programs behave similarly when extended by any further programs. This leads to the concept of strong equivalence, where programs P_1 and P_2 are strongly equivalent if and only if for any S, $P_1 \cup S$ is equivalent to (has the same answer sets as) $P_2 \cup S$. It is easy to get convinced that, whenever P_1 and P_2 are different formulation of a RASP program involving consumption and production of resources, their behaving equivalently in different contexts is of particular importance related to reliability in resource usage. Moreover, a designer might be able to evaluate, in terms of strong equivalence, different though analogous ways of producing certain resources, and thus be able to choose one rather than the other in terms of suitable criteria.

In order to extend the notion of strong equivalence to RASP, we need to reformulate the semantics of RASP programs so as to be similar to that of plain ASP programs. This is done in Sect. 2 after an introduction to RASP provided in Sect. 1. After that, the definition of strong equivalence developed in [9] can be fairly easily extended to RASP (Sect. 3). However, it turns out that RASP programs behave quite differently from ASP programs as far as strong equivalence is concerned, as interferences in resource usage among rules of P_1, P_2 and S can easily arise. Then, we will argue (Sect. 4) that a significant notion of strong equivalence for RASP programs requires to state some constraints on S. So done, strong equivalence becomes a more effective notion in the RASP context. Finally, in Sect. 5 we conclude and outline some possible future directions. Proofs of theorems can be found in Appendix A. We assume a reader to be familiar with both ASP and strong equivalence. The reader may refer for the former to [10] and to the references therein, and for the latter to [11].

1 Background on RASP

RASP [5, 6, 7] is an extension of the ASP framework obtained by explicitly introducing the notion of *resource*. It supports both formalization and quantitative reasoning on consumption and production of amounts of resources. These are modeled by *amount-atoms* of the form $q\!:\!a$, where q represents a specific type of resource and a denotes the corresponding amount. Resources can be produced or consumed (or declared available from the beginning).

The processes that transform some amounts of resources into other resources are specified by *r-rules*, for instance, as in this simple example:

$$computer\!:\!1 \leftarrow cpu\!:\!1, hd\!:\!2, motherboard\!:\!1, ram\_module\!:\!2.$$

where we model the fact that an instance of the resource *computer* can be obtained by "consuming" some other resources, in the indicated amounts.

In their most general form, r-rules may involve plain ASP literals together with amount-atoms. Semantics for RASP programs is given by combining stable model semantics with a notion of *allocation*. While stable models are used to deal with usual ASP literals, allocations are exploited to take care of amounts and resources. Intuitively, an allocation assigns to each amount-atom a (possibly null) quantity. Quantities are interpreted in an auxiliary algebraic structure that supports comparisons and operations on amounts. Thus, one has to choose a collection Q of *quantities*, the operations to combine and compare quantities, and a mapping that associates quantities to amount-symbols. Admissible allocations are those satisfying, for all resources, the requirement that one can consume only what has been produced. Alternative allocations might be possible. They correspond to different ways of using the same resources. A simple natural choice for Q is the set of integer numbers. In all the examples proposed in the rest of the paper, we implicitly make this choice.

Syntax and semantics of RASP are presented in [5]. Syntax is summarized below, semantics is reported shortly in Appendix B. Various extensions, presented in [5] and in [6, 7] (which discuss preferences and complex preferences in RASP) are not considered here. An implementation of RASP is discussed in [7] and is available at `http://www.dmi.unipg.it/~formis/raspberry/`.

RASP syntax is based upon partitioning the symbols of the underlying language into program symbols and resource symbols. Precisely, let $\langle \Pi, \mathcal{C}, \mathcal{V} \rangle$ be a structure where $\Pi = \Pi_{\mathcal{P}} \cup \Pi_{\mathcal{R}}$ is a set of predicate symbols such that $\Pi_{\mathcal{P}} \cap \Pi_{\mathcal{R}} = \emptyset$, $\mathcal{C} = \mathcal{C}_{\mathcal{P}} \cup \mathcal{C}_{\mathcal{R}}$ is a set of constant symbols such that $\mathcal{C}_{\mathcal{P}} \cap \mathcal{C}_{\mathcal{R}} = \emptyset$, and $\mathcal{V}$ is a set of variable symbols. The elements of $\mathcal{C}_{\mathcal{R}}$ are said *amount-symbols*, while the elements of $\Pi_{\mathcal{R}}$ are said *resource-predicates*. A *program-term* is either a variable or a constant symbol. An *amount-term* is either a variable or an amount-symbol.

The second step is that of introducing amount-atoms in addition to plain ASP atoms, called program-atoms. Let $\mathcal{A}(X, Y)$ denote the collection of all atoms of the form $p(t_1, \ldots, t_n)$, with $p \in X$ and $\{t_1, \ldots, t_n\} \subseteq Y$. Then, a *program-atom* is an element of $\mathcal{A}(\Pi_{\mathcal{P}}, \mathcal{C} \cup \mathcal{V})$. An *amount-atom* is an expression of the form $q{:}a$ where $q \in \Pi_{\mathcal{R}} \cup \mathcal{A}(\Pi_{\mathcal{R}}, \mathcal{C} \cup \mathcal{V})$ and a is an amount-term. Let $\tau_{\mathcal{R}} = \Pi_{\mathcal{R}} \cup \mathcal{A}(\Pi_{\mathcal{R}}, \mathcal{C})$. We call elements of τ_R *resource-symbols*.

Expressions such as $p(X){:}V$ where V, X are variable symbols are allowed, as resources can be either directly specified as constants or derived. Notice that the set of variables is not partitioned, as the same variable may occur both as a program term and as an amount-term. *Ground* amount- or program-atoms contain no variables. As usual, a *program-literal* L is a program-atom A or the negation $not\, A$ of a program-atom (intended as negation-as-failure).[4] If $L = A$ (resp., $L = not\, A$) then $\overline{L}$ denotes $not\, A$ (resp., A). A *resource-literal* is either a program-literal or an amount-atom. Notice that, for reasons discussed in [5], we do not admit negation of amount-atoms.

Finally, we distinguish between plain rules and rules that involve amount-atoms. In particular, a *program-rule* is defined as a plain ASP rule. Besides program-rules we introduce resource-rules which differ from program rules (which are usual ASP rules) in that they may contain amount-atoms. A *resource-proper-rule* has the form $H \leftarrow B_1, \ldots, B_k$. where $B_1, \ldots, B_k$, are resource-literals and H is either a program-atom or a (non-empty) list of amount-atoms. If H is an amount-atom of the form $q{:}a$ where a is a constant and the body is empty then the rule is called a *resource-fact*. According to the definition then, the amount of an initially available resource has to be explicitly stated. As usual, we often denote the resource-fact $H \leftarrow$ simply by writing H.

In general, we admit several amount-atoms in the head of a rule where the case in which a rule γ has an empty head is admitted only if γ is a program-rule (i.e., γ is an ASP *constraint*). The list of amount-atoms composing the head of

---

[4] In this paper we will only deal with negation-as-failure. Nevertheless, classical negation of program literals could be used in RASP programs and treated as usually done in ASP.

an resource-rule has to be understood conjunctively, i.e., as a collection of those resources that are all produced at the same time by *firing*, i.e. applying, the rule.

A *resource-rule* (*r-rule*, for short) can be either a resource-proper-rule or a resource-fact. A RASP program may involve both program rules and resource-rules, i.e., a *RASP-rule* (*rule*, for short) γ is either a program-rule or a resource-rule and a RASP program (*r-program*) is a finite set of RASP-rules.

The ground version (or "grounding") of an r-program P is the set of all ground instances of rules of P, obtained through ground substitutions over the constants occurring in P. As customary, in what follows we will implicitly refer to the ground version of P.

Intuitively, an interpretation of P is an answer set whenever it satisfies all the program rules in P and all the fired r-rules (in the usual way) as concerns their program-literals, and all consumed amounts either were available from resource-facts or have been produced by rule firings.

Example 1. Below is an example of RASP program.

$g{:}2 \leftarrow q{:}4.$
$f{:}3 \leftarrow q{:}2.$
$h{:}1 \leftarrow q{:}1.$
$q{:}4.$

This program has the following answer sets. The answer set $\{q{:}4, g{:}2, -q{:}4\}$, where we employ (consume) resource $q{:}4$ (as indicated by the notation $-q{:}4$) to *fire* (i.e., apply) the first rule and produce $g{:}2$, with no remainder (the full available amount of q is consumed). The answer set $\{q{:}4, f{:}3, h{:}1, -q{:}3\}$, where we employ (consume) part of resource $q{:}4$ (as indicated by the notation $-q{:}3$) to fire the second and third rule and produce $f{:}3$ and $h{:}1$. In this case, as we do not have consumed the full amount of q, there is a remainder $q{:}1$ that might have been potentially used by other rules. We cannot produce $g{:}2$ together with $f{:}3$ and/or $h{:}1$ because the available quantity of q is not sufficient. But, we also have the answer sets $\{q{:}4\}$, $\{q{:}4, f{:}3, -q{:}2\}$ and $\{q{:}4, h{:}1, -q{:}1\}$ where all of part of resource $q{:}4$, though available, is left unconsumed.

Notice that the given program does not involve negation. In plain ASP we may have several answer sets only if the program involves negation, and, in particular, cycles on negation. In RASP, we may have several answer sets also in positive (i.e., definite) programs because of different possible allocations of available resources.

Notice also that in the present setting each rule can be applied only once, i.e, we can't for instance use the third rule several times to produce several items of $h{:}1$ according to the available qs. Actually, multiple "firings" of rules can be allowed by a suitable specification but we do not consider this extension here. However, a rule might in principle occur more than once in a program: in this case, each "copy" of the rule can be separately applied.

In [5] we introduce *politics* for resource usage, by extending the semantics so as to allow a programmer to state, for each rule, if the firing is either optional or mandatory. In the rest of this paper, to simplify the discussion, we make the assumption that the firing of rules is mandatory. In the above example, this politics excludes the answer sets $\{q{:}4\}$, $\{q{:}4, f{:}3, -q{:}2\}$ and $\{q{:}4, h{:}1, -q{:}1\}$. In the next section, we introduce a version of the semantics of RASP close to the one originally defined for plain ASP, i.e., in terms of a reduct (for ASP, the so-called Gelfond-Lifschitz reduct, or GL-reduct, introduced in [1]).

2 Reduct-based RASP Semantics

In order to extend the notion of strong equivalence to RASP, it is useful to devise a semantic definition for RASP as close as possible to the standard answer set semantics as defined in [1]. This will make it easier to extend to RASP the definitions and proofs provided in [9] for plain ASP.

To our aim, we exploit the definition of RASP semantics discussed in Appendix B. It is based upon generating, from a ground r-program, a version where resource-predicates occurring in bodies of rules are associated to the rule where they occur. For the sake of simplicity and without loss of generality, we assume that a resource predicate, say q, may occur more than once in the body of the same rule only if the amounts are different. It may occur instead with either the same or different amounts in the head of several rules. Below, let a "program" be a RASP program. The answer sets for program obtained with the formulation proposed in this section are called *reduct-answer sets* (for short, *r-answer sets*).

In order to cope with different instances of the same amount-atom, say $q{:}a$, occurring in the body of different rules, at this stage we "standardize apart" these occurrences, according to the following definition.

Definition 1. *Let P be a ground r-program, and let $\gamma_1, ..., \gamma_k$ be the rules in P containing amount-atoms in their body. The standardized-apart version P_s of P is obtained from P by renaming each amount-atom $q{:}a$ in the body of γ_j, $j \leqslant k$ as $q^j{:}a$. The q^j's are called the standardized-apart versions of q, or in general standardized-apart resource-predicates.*

Example 2. Let P be the following program.

$g{:}2 \leftarrow q{:}4.$ $\quad$ $a \leftarrow not\ b.$
$p{:}3 \leftarrow q{:}3, a.$ $\quad$ $b \leftarrow not\ a.$
$d{:}1 \leftarrow q{:}3, b.$ $\quad$ $c.$
$q{:}6 \leftarrow c.$

The standardized-apart version P_s of P is as follows.

$g{:}2 \leftarrow q^1{:}4.$ $\quad$ $a \leftarrow not\ b.$
$p{:}3 \leftarrow q^2{:}3, a.$ $\quad$ $b \leftarrow not\ a.$
$d{:}1 \leftarrow q^3{:}3, b.$ $\quad$ $c.$
$q{:}6 \leftarrow c.$

We let $\mathcal{A}_{P_s}$ be the set of all atoms (both program-atoms and amount-atoms) that can be built from predicate and constant symbols occurring in P_s.

Definition 2. *A candidate reduct-interpretation* $\mathcal{I}_{P_s}$ *for* P_s *is any multiset obtained from a subset of* $\mathcal{A}_{P_s}$.

Referring to Example 2, among the possible candidate interpretations are, e.g., $I_1 = \{p{:}3, q{:}6, q^2{:}3, a, c\}$ and $I_2 = \{p{:}3, q{:}6, q^1{:}4, q^3{:}3, a, c\}$. Standardized-apart amount-atoms occurring in a candidate r-interpretation represent resources that are *consumed*. Plain amount-atoms represent resources that have been produced, or were available from the beginning. For a candidate r-interpretation to be an admissible r-interpretation (or, simply, r-interpretation), consumption has not to exceed production.

Definition 3. *Given multi-set of atoms* S *and resource-predicate* q *(possibly standardized-apart) occurring in* S*, let* $f(S)(q)$ *be an amount-symbol obtained by summing the quantities related to the occurrences of* q*. If* S *contains* $q{:}a_1$, $\ldots$, $q{:}a_k$ *(or, respectively,* $q^j{:}a_1$, $\ldots$, $q^j{:}a_k$*) and* $a = a_1 + \ldots + a_k$ *we will have* $f(S)(q) = a$ *(or, respectively,* $f(S)(q^j) = a$*).*

Definition 4. *A candidate reduct-interpretation* $\mathcal{I}_{P_s}$ *is a reduct-interpretation (for short r-interpretation) if for every resource-predicate* q *occurring in* $\mathcal{I}_{P_s}$*, taken all its standardized-apart versions* $q^{j_1}, \ldots, q^{j_h}$*,* $h \geq 0$*, also occurring in* $\mathcal{I}_{P_s}$*, we have* $f(\mathcal{I}_{P_s})(q) \geq \sum_{i=1}^{h} f(\mathcal{I}_{P_s})(q^{j_i})$

Referring again to Example 2, it is easy to see that I_1 is an r-interpretation, as 6 items of q are produced and just 3 are consumed, while I_2 is not, as 6 items of q are produced but 7 are supposed to be consumed.

We now establish whether an r-interpretation $\mathcal{I}_{P_s}$ is an r-answer set for P_s, and we will then reconstruct from it an r-answer set for P. To this aim, we introduce the following extension to the Gelfond-Lifschitz reduct.

Definition 5 (RASP-reduct). *Given a reduct-interpretation* $\mathcal{I}_{P_s}$ *for the standardized-apart version* P_s *of RASP program* P*, the RASP-reduct* $cfp(P_s, \mathcal{I}_{P_s})$ *is a RASP program obtained as follows.*

1. *For every standardized-apart amount-atom* $A \in \mathcal{I}_{P_s}$*, add* A *to* P_s *as a fact, obtaining* $P_s^+(\mathcal{I}_{P_s})$*;*
2. *Compute the GL-reduct of* $P_s^+(\mathcal{I}_{P_s})$.

Let $LM(T)$ be the Least Herbrand Model of theory T. In case T is a RASP program, in computing $LM(T)$ amount-atoms are treated as plain atoms. We may state the main definition:

Definition 6. *Given reduct-interpretation* $\mathcal{I}_{P_s}$ *for the standardized-apart version* P_s *of RASP program* P*,* $\mathcal{I}_{P_s}$ *is a reduct-answer set (r-answer set) of* P_s *if* $\mathcal{I}_{P_s} = LM(cfp(P_s, \mathcal{I}_{P_s}))$

Referring again to Example 2, the r-answer sets of P_s are:
$M_1 = \{q{:}6, g{:}2, q^1{:}4, a, c\}$, $M_2 = \{q{:}6, g{:}2, q^1{:}4, b, c\}$,
$M_3 = \{q{:}6, p{:}3, q^2{:}3, a, c\}$, $M_4 = \{q{:}6, d{:}1, q^3{:}3, b, c\}$.

Notice that M_1 and M_2 have the same resource consumption and production but from the even cycle on negation they choose a different alternative (a w.r.t.

b). Producing g:2 excludes being able to produce p:3 or d:1 respectively, because the remaining quantity of q is not sufficient. Instead of producing g:2, one can produce either p:3 (answer set M_3) if choosing the alternative a or d:1 (answer set M_4) if choosing the alternative b.

We now have to prove the equivalence of the above-proposed semantic formulation with the one of [5], summarized in Appendix B. In the Appendix, we report the notion of RASP answer set, and, in Definition 15, the equivalent notion of admissible answer set of an adapted RASP program. We will prove that there is a bijection between the set of the r-answer sets of the standardized-apart version P_s of RASP program P and the set of the admissible answer sets of the corresponding adapted program. It follows that there exists a bijection between the set of RASP answer sets of ground RASP program P and the set of r-answer sets of the standardized-apart version P_s of P.

In order to compare r-answer sets with admissible answer sets we have to make their form compatible. In particular, in Definition 14 of Appendix B we consider interpretations of an adapted program obtained from the standardized-apart program P_s augmented by adding, for each q^j:a, an even cycle involving q^j:a and the fresh atom no_q^j:a. These interpretations will thus contain either q^j:a or no_q^j:a to signify availability or, respectively, unavailability of this resource. Below we transform an r-interpretation into this form.

Definition 7. *Given an r-interpretation $\mathcal{I}_{P_s}$ for the standardized-apart version P_s of RASP program P, $ad(\mathcal{I}_{P_s})$ is obtained from $\mathcal{I}_{P_s}$ by adding no_q^j:a for every amount symbol q^j:a which occurs in P_s but not in $\mathcal{I}_{P_s}$.*

We can now state the result we where looking for:

Theorem 1. *Given an r-interpretation $\mathcal{I}_{P_s}$ for the standardized-apart version P_s of RASP program P, $\mathcal{I}_{P_s}$ is an r-answer set of P_s if and only if $ad(\mathcal{I}_{P_s})$ is an admissible answer set of the adapted program corresponding to P_s.*

Lemma 1. *There exists a bijection between the set of RASP answer sets of ground RASP program P and the set of r-answer sets of the standardized-apart version P_s of P.*

At this stage, as we wish to obtain r-answer sets of P, we can get a more compact form by getting the global consumed/produced quantity of each resource q.

Definition 8. *Given an r-answer set $\mathcal{I}_{P_s}$ of P_s, an r-answer set $\mathcal{M}^R_P$ of P is obtained as follows. For each resource-predicate q occurring in $\mathcal{I}_{P_s}$:*
(i) replace its occurrences q:b_1, ..., q:b_s, $s > 0$, with q:b, for $b = b_1 + \ldots + b_s$;
(ii) replace its standardized-apart occurrences q^{j_1}:a_1, ..., q^{j_k}:a_k, $k \geq 0$, with $-q$:a, for $a = a_1 + \ldots + a_k$.

By some abuse of notation, we will interchangeably mention r-answer set of P or of P_s. Referring to Example 2, the r-answer sets of P are:
$M_1 = \{q{:}6, g{:}2, -q{:}4, a, c\}$, $M_2 = \{q{:}6, g{:}2, -q{:}4, b, c\}$,
$M_3 = \{q{:}6, p{:}3, -q{:}3, c\}$, $M_4 = \{q{:}6, d{:}1, -q{:}3, b, c\}$.

Notice that the above notation does not explicitly report about what is left. Actually, given each r-answer set we can establish which resources we have "in our hands" after the production/consumption process by computing the difference, for each resource, between what has been produced and what has been consumed. For instance, in M_1 and M_2 we are left with q:2 and g:2, in M_3 with q:3 and p:3, and in M_4 with q:3 and d:1.

3 Strong Equivalence of RASP programs

In this section, we extend the standard notion of strong equivalence to RASP programs, taking as a basis the definitions that can be found in [9], which provides a characterization of strong equivalence of ground programs in terms of the propositional logic of here-and-there (HT-logic). We remind the reader that the logic of here-and-there is an intermediate logic between intuitionistic logic and classical logic. Like intuitionistic logic it can be semantically characterized by Kripke models, in particular using just two worlds, namely *here* and *there*, assuming that the *here* world is ordered before the *there* world. Accordingly, interpretations (HT-interpretations) are pairs (X, Y) of sets of atoms from given language L, such that $X \subseteq Y$. An HT-interpretation is total if $X = Y$. The intuition is that atoms in X (the *here* part) are considered to be true, atoms not in Y (the *there* part) are considered to be false, while the remaining atoms (from $Y \setminus X$) are undefined. A total HT-interpretation (Y, Y) is called an equilibrium model of a theory T, iff $(Y, Y) \models T$ and for all HT-interpretations (X, Y), such that $X \subset Y$, it holds that $(X, Y) \not\models T$. For an answer set program P, it turns out that an interpretation Y is an answer set of P iff (Y, Y) is an equilibrium model of P when reinterpreted as an HT-theory.

We take as a basis the standardized-apart version P_s of RASP program P. To account for resource production and consumption, HT-logic must be extended as follows. The set of atoms must be augmented to admit amount-atoms, involving both plain and standardized-apart resource predicates. Like in the RASP semantics, we take for given the choice of an algebraic structure to represent amounts and support operations on them.

The satisfaction relation of HT-logic between an interpretation $I = \langle I^H, I^T \rangle$ and a formula F must be augmented by adding the following two new axioms (where w is the world, that can be either *here* or *there*).

The first new axiom "distributes" the available quantity of a resource q to the formulas that need to use it. Notice that the total quantity of q can be produced in various items, but the condition is that the quantity which is distributed cannot exceed production.

AR-1
$\langle I^H, I^T, w \rangle \models q^{j_1}{:}a_1 \wedge \ldots \wedge q^{j_k}{:}a_k$ where $k > 0$ and each $j_i > 0$ if
$\langle I^H, I^T, w \rangle \models q{:}b_1 \wedge \ldots \wedge q{:}b_s$, $s > 0$, and $b_1 + \ldots + b_s \geq a_1 + \ldots + a_k$

Notice that, for every HT-interpretation $\langle I_s^H, I_s^T \rangle$ of P_s, AR-1 ensures that the sets I_s^H and I_s^T are, in the terminology of previous section (Definition 4), r-interpretations of P_s.

The second new axiom accounts for the fact that, given the final total quantity, the "fragments" in which a resource amount is produced do not matter. To this aim, for each resource-predicate q we provisionally introduce a fresh corresponding resource-predicate q^T.

AR-2
$\langle I^H, I^T, w\rangle \models q^T{:}b$ if $\langle I^H, I^T, w\rangle \models q{:}a_1 \wedge \ldots \wedge q{:}a_k$ and
$\langle I^H, I^T, w\rangle \not\models q{:}a,\ a \neq a_1 \wedge \ldots \wedge a \neq a_k$, where $k > 0$ and $b = a_1 + \ldots + a_k$

Notice that AR-2 is able to "compute" the total produced quantity b of each resource q. In order to deal with different though equivalent production patterns that may occur in different RASP programs, we keep in HT-interpretations only the total quantities of produced resources.

Definition 9. *Given an HT-interpretation $\langle I_s^H, I_s^T\rangle$ of P_s, its* normalized version *is obtained by replacing in both I_s^H and I_s^T for each resource-predicate q the set of atoms $q^T{:}b, q{:}a_1, \ldots, q{:}a_k$ with $q : b$.*

In what follows, by some abuse of notation, by "HT-interpretation" we mean its normalized version (the same for HT-models). This stated, it is not difficult to suitably extend to RASP the results introduced in [9]. We have the following lemma and we can state the main theorem:

Lemma 2. *For any RASP program P and any set I of atoms, the HT-interpretation $\langle I, I\rangle$ is an equilibrium model of P iff I is an r-answer set of P_s.*

Theorem 2. *For any RASP programs P_1 and P_2, and for every RASP program S, $P_1 \cup S$ has the same answer sets of $P_2 \cup S$, i.e., P_1 is strongly equivalent to P_2, if and only if their standardized-apart versions P_{1s} and P_{2s} are equivalent in the extended logic of Here-and-There, i.e., have the same HT-models.*

As we have seen before, axiom AR-2 and normalized versions of HT-interpretations account for different production patterns. Thus, two RASP programs are candidates to be equivalent whenever the same total quantity of each resource q is produced. Instead, consumption must be performed in exactly the same way, which, as we will see, is a quite strong limitation. In the rest of this section in fact, we discuss a number of examples to illustrate the meaning of strong equivalence for RASP programs, and emphasize the problems which arise. We will then address these problems in the subsequent section.

Example 3. The following two RASP programs are strongly equivalent.

P_1	P_2
$q{:}6 \leftarrow not\ c.$	$q{:}3 \leftarrow not\ c.$
	$q{:}3 \leftarrow not\ c.$

Their unique equilibrium model is $\langle I, I\rangle$ with $I = \{q : 6\}$ which is the unique r-answer set of both programs. The only difference between the two programs is that the same amount of resource q is produced in P_1 all in once, and in P_2 in two parts. This difference is taken into account by axiom AR-2.

However, in the terms illustrated up to now, it is very difficult for RASP programs to be not only strongly equivalent, but even simply equivalent, as demonstrated by the following examples.

Example 4. The following two standardized-apart RASP programs are not even equivalent, despite the fact that they produce and consume the same resources, though in a different way.

P_1
$q{:}1 \leftarrow p^1{:}2.$ $\quad p{:}6.$
$r{:}1 \leftarrow p^2{:}3.$ $\quad g{:}1.$
Unique answer set: $\{g{:}1, p{:}6, q{:}1, r{:}1, p^1{:}2, p^2{:}3\}$.

P_2
$q{:}1 \leftarrow p^1{:}3.$ $\quad p{:}6.$
$r{:}1 \leftarrow p^2{:}2.$ $\quad g{:}1.$
Unique answer set: $\{g{:}1, p{:}6, q{:}1, r{:}1, p^1{:}3, p^2{:}2\}$.

By summing quantities like in Definition 8 in previous section, one would obtain the same unique r-answer set for both, namely $\{g{:}1, p{:}6, q{:}1, r{:}1, -p{:}5\}$. Even considering the r-answer set, the two programs are not strongly equivalent, as we can see if one adds, e.g., as third rule $g{:}2 \leftarrow p{:}3$ (that, standardized-apart, becomes $g{:}2 \leftarrow p^3{:}3$). The reason is that the additional rule "competes" with the original ones for the use of amounts of resource $p{:}6$. Different choices for employing resource $p{:}6$ become thus possible. In fact, for the augmented former program the answer sets would be:
$M_1 = \{g{:}1, p{:}6, q{:}1, r{:}1, p^1{:}2, p^2{:}3\}$ $M_2 = \{g{:}1, p{:}6, q{:}1, g{:}2, p^1{:}2, p^3{:}3\}$ and
$M_3 = \{g{:}1, p{:}6, r{:}1, g{:}2, p^2{:}3, p^3{:}3\}$
For the latter program they would be instead:
$N_1 = \{g{:}1, p{:}6, q{:}1, r{:}1, p^1{:}3, p^2{:}2\}$ $N_2 = \{g{:}1, p{:}6, q{:}1, g{:}2, p^1{:}3, p^3{:}3\}$ and
$N_3 = \{g{:}1, p{:}6, r{:}1, g{:}2, p^2{:}2, p^3{:}3\}$

Example 5. The following two standardized-apart RASP programs are, differently from what would happen in plain ASP, not even equivalent. In fact, in the former program there is a number of plainly available resources, while in the latter producing $q{:}4$ requires consuming $p{:}2$.

P_1
$q{:}4.$ $\quad p{:}2.$
$r{:}4.$ $\quad c.$
Unique answer set: $\{c, q{:}4, p{:}2, r{:}4\}$

P_2
$q{:}4 \leftarrow p^1{:}2.$ $\quad p{:}2.$
$r{:}4.$ $\quad c.$
Unique answer set: $\{c, q{:}4, p{:}2, r{:}4, p^1{:}2\}$

However, in both examples one may notice that the two programs are equivalent w.r.t. what is produced, and differ w.r.t. resources that are consumed. A conceptual tool to recognize some kind of equivalence of the above programs is in order, as a designer would be enabled to assess their similarity and choose the way of production deemed more appropriate in the application at hand. In the next section we will propose a weaker notion of equivalence and strong equivalence than those introduced so far.

4 Strong Equivalence of RASP programs revisited

Below we introduce a compact form of HT-models where produced and consumed resources occur in their total quantities, similarly to r-answer sets introduced in Definition 8.

Definition 10. *Given an HT-model $\langle I_s^H, I_s^T \rangle$ of P_s, a* compact *HT-model (c-HT-model) $\langle I^H, I^T \rangle$ of P is obtained by replacing, for each resource-predicate q occurring in $\langle I_s^H, I_s^T \rangle$, its standardized-apart occurrences $q^{j_1}{:}a_1, \ldots, q^{j_k}{:}a_k$, $k \geq 0$, with $-q{:}a$, where $a = a_1 + \ldots + a_k$.*

The following definition make a further simplification by eliminating consumed quantities from r-answers sets and c-HT-models.

Definition 11. *Given an r-answer set A of P or given a c-HT-model $\langle I_c^H, I_c^T \rangle$ of P_s, a* production *answer set (p-answer set) A' of P (resp., a* production *HT-model, or p-HT-model) $\langle I^H, I^T \rangle$ of P_s) is obtained as by removing from A (resp., from I_c^H and I_c^T) all atoms of the form $-q{:}a$ for any q and a.*

In Example 4, the unique equilibrium c-HT-model of both programs is $\langle I, I \rangle$ with $I = \{g{:}1, p{:}6, q{:}1, r{:}1, -p{:}5\}$ where I is the unique r-answer set. The corresponding p-answer set is $I' = \{g{:}1, p{:}6, q{:}1, r{:}1\}$, and the unique equilibrium p-HT-model is $\langle I', I' \rangle$.

In Example 5, the unique r-answer set of P_1 is $M = \{c, q{:}4, p{:}2, r{:}4\}$, which coincides with the p-answer set. The unique equilibrium c- and p-HT-model is $\langle M, M \rangle$. For P_2, the unique r-answer set is $N = \{c, q{:}4, p{:}2, r{:}4, -p{:}2\}$, and the unique equilibrium c-HT-model is $\langle N, N \rangle$. The corresponding p-answer set is $N' = \{c, q{:}4, p{:}2, r{:}4\}$, and the unique equilibrium p-HT-model is $\langle N', N' \rangle$. Thus, in both Example 4 and Example 5 the two given programs are "equivalent"w.r.t. equilibrium p-HT-models (or, equivalently, p-answer sets). We formalize this notion of equivalence below.

Definition 12. *Two RASP theories (standardized-apart programs) are* equivalent on production *(p-equivalent) if their p-HT-models (and, consequently, their p-answer sets) coincide and are* strongly equivalent on production *(p-strongly equivalent) if, after adding whatever RASP theory S to both, they are still equivalent on production.*

This definition much enlarges the set of RASP programs that can be considered to be equivalent. However, strong equivalence remains problematic.

Example 6. Consider the two programs of Example 5, that are equivalent on production. Assume to add rule: $r{:}1 \leftarrow p^4{:}3$. The c-HT-models and the p-HT-models remain unchanged, and thus the two programs are still p-equivalent. In fact the added rule cannot fire, not being available the needed quantity of p. Assume instead to add rule $r{:}1 \leftarrow p^4{:}2$, which is able to exploit the available resource amount $p{:}2$. In this case, for the augmented P_1 we get the unique r-answer set $\{c, q{:}4, p{:}2, r{:}1, -p{:}2\}$, but for the augmented P_2 the new rule "competes" with pre-existing ones on the use of $p{:}2$: thus, we get the two r-answer sets $\{c, p{:}2, r{:}4, r{:}1, -p{:}2\}$ and $\{c, q{:}4, p{:}2, r{:}4, -p{:}2\}$. Consequently, the updated programs are not p-equivalent.

We may notice that the problems that we have discussed arise when S is a *proper RASP program*, i.e., a RASP program containing amount-atoms. In order to make strong equivalence possible, we may introduce the constraint that whenever the addition S is a proper RASP program, it does not compete on resources with the original program. This appears quite reasonable: in fact, if one intends to enlarge a production/consumption process, preservation of existing processes should be guaranteed. If not, it should be clear that one obtains a different process, with hardly predictable properties.

Definition 13. *A* rational addition *S to a RASP program P is RASP program such that in program $P \cup S$ rules of S do not consume resources produced by rules of P.*

It is easy to get convinced that:

Proposition 1. *RASP proper program S is a rational addition if and only if one of the following conditions hold.*

1. *Resources consumed in S are not available in P.*
2. *Resources consumed in S are produced in S.*
3. *Resources consumed in S are produced in $P \cup S$.*
4. *S does not consume resources.*

Referring to Example 5: rule $r{:}1 \leftarrow p^4{:}3$ is a rational addition of kind (1); RASP proper program $r{:}1 \leftarrow p^4{:}3.\ p{:}5.$ is a rational addition of kind (2); RASP proper program $r{:}1 \leftarrow p^4{:}3.\ p{:}5 \leftarrow c.$ is a rational addition of kind (3); rule $r{:}1 \leftarrow c$ is a rational addition of kind (4).

If we restrict p-strong equivalence to p-strong equivalence w.r.t. rational addition, then the notion becomes much more usable in practical cases. In particular for instance, programs P_1 and P_2 of Examples 4 and 5 are p-strongly equivalent w.r.t. rational addition. Notice however that p-strong equivalence between two programs does not guarantee that we are left with the same resources in the two cases. A subject of future work will be that of studying other forms of strong equivalence, e.g. w.r.t. what is left or more particularly what is left for a certain set of resources.

5 Conclusions and Future Directions

In this paper, we have extended the notion of strong equivalence from answer set programs to RASP programs. We have seen that this notion takes a quite peculiar flavor in RASP, where strong equivalence (apart from trivial cases) can be ensured at the condition of imposing some requirements on the theory which is added to given one. In fact, resource usages in the two components may interact in non-trivial ways. Nonetheless, strong equivalence may help a designer to reason about different though equivalent or partially equivalent processes.

As future direction, we may notice that, in the RASP context, an extension of strong equivalence, i.e., synonymy [12], can find an interesting application. Synonymy extends strong equivalence in the sense that two theories corresponding to different descriptions of a piece of knowledge may be equivalent even if they are expressed in different languages, if each is bijectively interpretable in the other answer set. Moreover, in a suitable sense, they can remain equivalent or synonymous when extended by the addition of new formulas.

In RASP, this may allow one to compare different processes formulated in seemingly different ways in order to establish whether they are “in essence” equivalent. Then, in practical applications one will be able to choose between the different formulations according to suitable parameters.

References

[1] Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R., Bowen, K., eds.: Proc. of the 5th Intl. Conference and Symposium on Logic Programming, The MIT Press (1988) 1070–1080

[2] Marek, V.W., Truszczyński, M. In: Stable logic programming - an alternative logic programming paradigm. Springer (1999) 375–398

[3] Baral, C.: Knowledge representation, reasoning and declarative problem solving. Cambridge University Press (2003)

[4] Anger, C., Schaub, T., Truszczyński, M.: ASPARAGUS – the Dagstuhl Initiative. ALP Newsletter **17**(3) (2004) See `http://asparagus.cs.uni-potsdam.de`.

[5] Costantini, S., Formisano, A.: Answer set programming with resources. Journal of Logic and Computation **20**(2) (2010) 533–571

[6] Costantini, S., Formisano, A.: Modeling preferences and conditional preferences on resource consumption and production in ASP. Journal of Algorithms in Cognition, Informatics and Logic **64**(1) (2009) 3–15

[7] Costantini, S., Formisano, A., Petturiti, D.: Extending and implementing RASP. Fundamenta Informaticae **105**(1-2) (2010) 1–33

[8] Pearce, D.: A new logical characterization of stable models and answer sets. In: Non-Monotonic Extensions of Logic Programming. Number 1216 in LNAI. Springer (1997) 55–70

[9] Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Transactions on Computational Logic **2** (2001) 526–541

[10] Gelfond, M.: Answer sets. In: Handbook of Knowledge Representation. Chapter 7. Elsevier (2007)

[11] Pearce, D., Valverde, A.: Quantified equilibrium logic and the first order logic of here-and-there. Technical report, Univ. Rey Juan Carlos (2006) available at http://www.satd.uma.es/matap/investig/tr/ma06_02.pdf.

[12] Pearce, D., Valverde, A.: Synonymous theories in answer set programming and equilibrium logic. Proc. of ECAI04, 16th Europ. Conf. on Art. Intell. (2004) 388–390

A Proofs of Theorems

Proof of Theorem 1

Proof. If part.
It is easy to see that the r-answer sets obtained by means of the RASP-reduct can equivalently be obtained by applying the GL-reduct to a version of P_s where, instead of asserting the $q^j\!:\!a$'s as facts, one would have added to P_s for each of them an even cycle as specified in Definition 14, thus obtaining the adapted program corresponding to P_s. In fact, one would get exactly $ad(\mathcal{I}_{P_s})$ that, by Definition 4, obeys the condition of Definition 15.
Only-if part.
If one removes the $no\_q^j\!:\!a$'s from $ad(\mathcal{I}_{P_s})$ one obtains a candidate r-interpretation $\mathcal{I}_{P_s}$ for P_s that, being an admissible answer set for the adapted program corresponding to P_s, is an r-answer set for P_s because: (i) admissibility of $ad(\mathcal{I}_{P_s})$ guarantees that the condition specified in Definition 4 is fulfilled, and thus the candidate r-interpretation is actually an r-interpretation; (ii) except for the additional even cycles, rules of P_s and of the corresponding adapted program are the same; (iii) the RASP-reduct exploits the $q^j\!:\!a$'s present in an r-interpretation as though they had been generated by even cycles;

Proof of Theorem 2 (Main Theorem)

The proof is sketched, as it can be obtained as a variation of the proof provided in [9] for plain ASP program. Therefore, we will not repeat all the steps of their proof (that takes many pages), rather we emphasize the variations. Below, by "HT-logic" we take the extended HT-logic defined in Sect. 3.

First of all it is useful to extend Lemma 3 of [9] to RASP programs. To this aim, we need:
Fact 1: Given RASP program P, for every HT-model $\langle H, T\rangle$ of its standardized-apart version P_s both H and T are r-interpretations for P.

In fact, axiom AR-1 of extended HT-logic ensures that from a resource q available in a certain total quantity ("computed" by axiom AR-2 of the logic) one can obtain standardized-apart portions of that resource in an amount not exceeding the available one, as required by Def. 4 of Sect. 2. This allows us to restate Lemma 3 of [9] as Lemma 2, proved below. So done, we have all the elements to conclude by applying the proof of [9] to P_{1s} and P_{2s}.

Proof of Lemma 2

Proof. To prove this lemma we simply have to notice that, from Definitions 5 and 6, an r-interpretation I of P is r-answer set I of P_s iff it is an answer set of $P_s^+(I)$, obtained by adding to P_s the standardized-apart atoms occurring in I as facts. Then, we can apply the proof of [9] to $P_s^+(I)$ given that axiom AR-2 accounts for different production patterns.

B RASP Semantics

In this section we summarize the semantics of RASP. In [5], there are two formulations of the RASP semantics, the the first one more "abstract" and the second one more "practical", i.e., closer to standard ASP. In this context we focus on the latter, that was originally provided in order to evaluate RASP complexity (which b.t.w. is the same as plain ASP) and has proved also useful for defining strong equivalence.

In the more abstract semantic definition, an interpretation/answer set of a RASP program is defined as a couple $\langle I, \mu \rangle$ where I takes care of program atoms, and μ is a function which assigns quantities to all occurrences of resource symbols, where the balance must be positive, in the sense that only what has been produced (or was available from the beginning) can be consumed. In the more practical semantic formulation, resource-predicates occurring in the body of rules are standardized-apart, by substituting them by means of fresh resource-predicates. This kind of elaboration is formalized in Definition 1, seen earlier.

By considering ground amount-atoms as plain atoms, we can now add, for each standardized-apart version of an amount-atom, an even cycle which simulates this resource to be allocated to the r-rule where it occurs or not (we can add such an even cycle only if that resource can potentially be produced by some other r-rule).

Definition 14. *Let P be a (ground) r-program, and let P_s be the standardized-apart version of P. The* adapted program *P' for P is obtained by adding to P_s for each $q^j{:}a$ occurring in the body of some r-rule of P_s and such that $q{:}b$ (for some b) occurs in the head of some r-rule of P_s the following pair of rules (where $no\_q^j{:}a$ is a fresh atom):*

$$q^j{:}a \leftarrow not\ no\_q^j{:}a.$$
$$not\ no\_q^j{:}a \leftarrow not\ q^j{:}a.$$

We can thus obtain the answer sets of the adapted program P', that we call "classical answer sets" of P'.

Below we state that a classical answer set M of the adapted program is *admissible* if the resources that have been used does not exceed the resources that were available. Notice that for each resource-predicate q, the available quantity t_a is obtained by summing all amounts of atoms of the form $q{:}a$ (that in the adapted program are found in the head of rules), while the total consumed quantity is obtained by summing all amounts of their standardized-apart versions (that in the adapted program are found in the body of rules).

Definition 15. *Let P be a (ground) r-program and P' the corresponding adapted program P'. A classical answer set M of P' is* admissible *if the* resource balance relative to *M is non-negative, i.e.:*

$$\forall q \in \tau_{\mathcal{R}} \ \left(\sum \{\!\!\{ a \mid q{:}a \in M \}\!\!\}\right) - \left(\sum \{\!\!\{ a \mid q^j{:}a \in M \ for\ some\ j \}\!\!\}\right) \geqslant 0$$

In [5] we have formally stated the relationship between RASP answer sets of program P and admissible answer sets of the adapted program P'.

Informally, given an admissible answer set M of the adapted program P', we have to identify in M the two components. Let $\mathcal{P}(M)$ be obtained from M as the subset including the program-atoms only. The particular quantities-assignment function $\mu = \nu(M)$ is elicited from M by collecting the quantities associated to amount-atoms occurring in rules which are fired in M, i.e., such that the head is in M and the body literals are satisfied in M.

It is thus possible to state the equivalence of the two semantic formulations.

Theorem 3. *$\langle I, \mu \rangle$ is a RASP answer set of a ground r-program P iff $I = \mathcal{P}(M)$ and $\mu = \nu(M)$ where M is an admissible answer set of the adapted program P' for P.*